EXPRESS MAIL LABEL NO.: EL563155585US
DATE MAILED: September 8, 2000 **PATENT**

INVENTOR: Jeffrey B. LOTSPIECH
Stefan NUSSER
Paul R. RETTIG

SYSTEM AND METHOD FOR SECURE AUTHENTICATION OF EXTERNAL SOFTWARE MODULES PROVIDED BY THIRD PARTIES

PARTIAL WAIVER OF COPYRIGHT

All of the material in this patent application is subject to copyright protection under the copyright laws of the United States and of other countries. As of the first effective filing date of the present application, this material is protected as unpublished material. However, permission to copy this material is hereby granted to the extent that the copyright owner has no objection to the facsimile reproduction by anyone of the patent documentation or patent disclosure, as it appears in the United States Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

CROSS-REFERENCE TO RELATED APPLICATIONS

This patent application is related to U.S. Patent Application No. [Number not assigned yet], Attorney Docket Number ARC9-2000-0057-US1, filed on even date herewith, commonly assigned to the assignee hereof, and the entire disclosure of which is herein incorporated by reference.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to secure electronic delivery of content, such as music or

movies. More particularly, the invention relates to an improved method and apparatus for authenticating external modules.

2. The Prior Art

An important problem in protecting content is making sure that only compliant, authorized software modules are allowed to process the content on the user's computer. For example, the content owners might want to restrict the number of perfect digital copies that the end users can make from a single piece of purchased content. Or, they might want to allow a restricted "preview mode", where the content (like a song) is delivered for free, but only be allowed to play some limited number of times. Against this desire of the content owners, are armies of hackers who want to figure out how to patch the software modules to disable the restrictions.

Therefore, code authentication becomes a critical problem: determining whether a given software module is intact and compliant, or has been hacked. In the art, authentication is often treated as a cryptographic problem. The standard way to verify authentication is with digital signatures.

Further, even though the electronic industry is enamored with using digital signatures for code authentication problems, the use of signatures is in reality a poor substitute for a robust cryptographic solution for the following reasons. A digital signature resolves itself into a single yes/no decision, i.e., "did the signature match"? A single jump instruction is thereby generated, which, if patched, disables the signature checking. A hacker's job is facilitated by such a simplistic method. Another algorithm for authenticating data files includes the generation and use of what is known as a "signet". Signets are described in U.S. patents US 6,038,316 and US 5,978,482 assigned to International Business Machines Corporation. These patents are hereby incorporated by reference. Signets are utilized according to the following description to provide a method and system to distribute extrication functions to legitimate users of information that are (1) openly available, publicly computable, and computationally infeasible to invert; (2) use an extremely long decryption key; and (3) only short

communication from the authorization center occurs. An authorization center processor receives a user identifying signal value ni, and then responsively creates a corresponding authorization signal ai, called a "signet", since it shares some properties of a digital signature. The pair (ai, ni) is called a "signet pair". The extrication function operates on the signet pair to produce a key signal K that may be used to decrypt the digital information. The extrication function is publicly computable. However, it is computationally infeasible to determine how to create a new authorization signal aj for a corresponding user processor identifying signal value nj, or even to create any new valid signet pair wherein a "valid signet pair" is any pair (x, y) in which x is the result obtained by applying the authorization function, or computation, to y. An alternative method offers the feature that, although the signet pair is short, the key produced by the extrication function is long. The combination of having a long key K with the intractability of generating a new signet pair based on previous signet pairs and the extrication function, serves to de-motivate (i.e. by use of a long K) and incapacitate (i.e. the feature of intractability) those who might wish to attempt to become illegitimate or pirate authorization centers. So, IBM's digital signet technology (which generate thousands of bytes that are correct if and only if the signet matches the hash) is far superior to that of an ordinary signature in this application.

The attack that these hackers utilize is that of patching external modules with illicit functions. The hackers then distribute these hacked programs on the Internet. Users search these out and download them, because copying restrictions have been removed that the legitimate programs enforce. For example, last November, a program called “DeCSS” appeared on the Internet. It allowed users to copy DVD videos, something that legitimate DVD video players are forbidden (by license) to do. While DeCSS was an entire application, this invention envisions that it is not necessary for the hacker to provide an entire program. For example, he may be able to replace only a single module within a program to achieve the result he desires. This problem is exacerbated in the modern world, because programs are often comprised, in part, of external modules provided by manufacturers different than the one manufacturing the

program. In particular, a system optionally uses these external modules which are loaded by an application on demand. External modules are typically implemented as DLLs under WIN32 or shared object (SO) files on UNIX platforms and are provided by third party manufacturers.

An example for an interaction between an application and external modules is the EMMS, or Electronic Media Management System, end user application and its use of decoder modules. Based on the type of encoding of an album, a decoder module needs to be located and loaded. Once the module is loaded, its externalized functions are called in order to perform the decompression of the audio data.

This example also highlights the security risk of using external modules. Essentially, the application loads and executes untrusted code. In the above example, that code gets access to the encoded audio data. The existence of a published API (like ACM, an Audio Compression Manager published by Microsoft for modules that are transforming an audio stream for decoders) makes it particularly easy for an attacker to replace the external module with a different one which, for example, simply writes the compressed audio data to disk and thereby obtains an in-the-clear copy which could then be distributed on the Internet.

Any authentication technology used for authenticating external modules needs to take into account that the external module remains functional so that it can be used by other applications which do not require a high level of trust, or deploy their own authentication scheme. Consequently, the external module cannot be simply encrypted by the secure application since that would prevent any other application from using it. As shown by the example above, code in external modules should never be executed without first authenticating that module.

Therefore, what is desired is a more efficient method, apparatus and product for securing and verifying the authenticity of external software modules.

SUMMARY OF THE INVENTION

The preferred embodiment of the invention comprises an external module that

has been loaded into an entity's memory being transformed by two functions. These are namely, the STOMP function and the UNSTOMP function. One or both of these functions is based, in some way, on the actual code that is found in the legitimate version of the external module. In other words, the STOMP-UNSTOMP pair produces an external module that in the preferred embodiment works differently if even a single byte of code in the external module has been changed by an attacker.

Essentially, the STOMP and the UNSTOMP are paired. The STOMP transforms the external module and makes it temporarily unusable whilst conversely, the UNSTOMP repairs the damage and makes it workable again. Thus, if the module is not authentic, the pairing between the STOMP and UNSTOMP is broken, because at least one of them behaves differently. Therefore, a patched module from a hacker remains unusable since the STOMP and UNSTOMP transformations do not produce a working external module. Because of the STOMP and UNSTOMP technique, an application is secure because if an external module is free from tampering then the application executes normally. In the event that an illicitly patched external module is loaded then the application fails. In either case, no audio, video or information content is illegally copied because of the disablement of the external module by the STOMP-UNSTOMP procedure.

BRIEF DESCRIPTION OF THE FIGURES

The subject matter which is regarded as the invention is particularly pointed out and distinctly claimed in the claims at the conclusion of the specification. The foregoing and other features, and advantages of the invention will be apparent from the following detailed description taken in conjunction with the accompanying drawings.

FIG. 1 is a flow diagram (100) that describes the process of generating public/private key pairs as practiced in the invention.

FIG. 2 is a flow diagram (200) that describes the process of generating an authentication token for an external module.

FIG. 3 is a flow diagram (300) that describes the process of STOMPing an

external module for authentication purposes.

FIG. 4 is a flow diagram (400) that describes the process of UNSTOMPing an external module to ensure the authenticity of the external module.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

It is important to note that these embodiments are only examples of the many advantageous uses of the innovative teachings herein. In general, statements made in the specification of the present application do not necessarily limit any of the various claimed inventions. Moreover, some statements may apply to some inventive features but not to others. In general, unless otherwise indicated, singular elements may be in the plural and vice versa with no loss of generality.

In the drawing like numerals refer to like parts through several views.

Exemplary Embodiment

"\l 2 The scope of this invention comprises all methods of STOMPing and UNSTOMPing the external module. In the preferred embodiment, a function based on an RSA encryption is utilized to STOMP the external module, and a function based on digital signets is utilized to UNSTOMP the STOMPed external module. Digital signets are described in US patent numbers 6,038,316, and 5,978,482. The process comprises several steps including: 1) the generation of public/private key pairs, 2) the generation of an authentication token, 3) the STOMPing of an external module and 4) the UNSTOMPing of a STOMPed external module. First, a discussion of the generation of public/private key pairs.

FIG. 1 comprises a flow diagram (100) that describes the process of generating public/private key pairs as practiced in the invention. First, public and private components of a signet are generated (102). The signet's private component is made available to a code authentication authority (104) and is to be used by this authority to create authentication tokens for external modules. Then an RSA key pair is generated (106). The public RSA key is embedded in the verification code of the application (108).

Finally, the private key is made available to a code authentication authority (110) and is to be used by this authority to create authentication tokens for external modules. The signet's private component and the RSA private key are either transferred to the code authentication authority in a secure fashion or, better still, generated by the code authentication authority itself. The system depends on these private components being not shared with other entities. The code authentication authority is the entity that is trusted by all the other entities to be able to determine whether an external module is indeed implementing the desired functionality. Various delegation mechanisms are within the scope of this disclosure. These key generation steps comprise an initial setup phase that needs to be performed before any authentication tokens can be issued. Next, a description of the process for generating an authentication token is described.

FIG. 2 comprises a flow diagram (200) that describes the process of generating an authentication token for an external module. The process of generating a token for an external module involves the following steps. First, a certain number of random bytes (K) are generated (202) so that they are valid signet extrication data. Then K is encrypted (204) with an RSA private key. The encrypted K becomes part of the authentication (206) token. Finally, the signet is calculated (208) using the hash of the external module as input so that the signet upon successful extrication creates K. As a result, both of the following statements are true. Namely, K gets recreated at run time in two different ways: 1) once by means of the RSA calculation during the STOMP and than 2) as the signet extrication data during the UNSTOMP. If the module is modified, the hash is different and the signet extrication process creates a different K. As a consequence, the UNSTOMP does not reproduce the module correctly. Using the invention presented in this disclosure, the private components, namely, the RSA private key and the private signet component are used to calculate authentication tokens for each external module. Thus, the authentication token consists of the signet on the external module, and K encrypted with the RSA private key. In addition, only public information is placed in the secure application itself. After the process of generating

keys and tokens has completed the authentication process is ready for field use.

In particular, at run time, the verification code performs the steps as shown in FIG. 3 before the external module is accessed. FIG. 3 comprises a flow diagram (300) that describes the process of STOMPing an external module for authentication purposes. First, the external module is loaded (302) into memory. Then the STOMPing process begins by the embedded RSA public key being used to decrypt (304) the encrypted K that is part of the authentication token. That K generates (306) a stream of pseudo random bytes. Finally, these pseudo random bytes are XORed (308) with the module in memory that makes the module unusable. Once the STOMPing process has completed the external module may be validated using the UNSTOMPing procedure described below.

FIG. 4 comprises a flow diagram (400) that describes the process of UNSTOMPing an external module to ensure the authenticity of the external module. First, the signet extrication process begins (402) by using the cryptographic hash of the external module to extract data (404). Here, the cryptographic hash serves the role of the "user data" as described in the original signet patents. The extrication data which, if the external module has not been tampered with should be the same K, is used to generate (406) the same stream of pseudo-random bytes as previously described. However, the method used to produce the pseudo random byte stream is different from that described previously. Thus, the process produces the same stream of random bytes in two different ways at least one set of random bytes being dependent upon the hash. Then a generation technique employed here is that these bytes are XORed with the external module mapped in memory (408), thus undoing the STOMPing in the step above.

Finally, the verification code needs access to the external module and the corresponding authentication token in order to verify the authenticity of the external software module. If the external module verification succeeds, the external module is mapped into memory and restored to its original content. However, if the verification process fails, the external module consists of random bytes which when executed

undoubtedly cause an application error. No illicit patches are thereby allowed to effect the operation of a computer entity.

High Level Overview

The invention as described further comprises the following additional unique characteristics:

1. The secure hash algorithm used for the signet needs to take into account that the external module might have been relocated by the operating system's loader. One way to resolve this issue is by hashing the external module as a file on the hard drive as opposed to as an image in memory. In that case, of particular importance is the ability to get a strong association between the module mapped in memory and the source file name. The Windows family of operating systems provides a system call for obtaining that information, i.e., the GetModuleFilename(X) API.
2. All DLLs are loaded by the system loader to their preferred base address which is hardwired in the DLL module. Obviously collisions may occur if two DLLs are set up for the same base address. In that case, the loader performs a DLL relocation, meaning that one of the modules gets moved to a different base address. As part of that relocation, certain instructions in the DLL (like absolute references) are modified by the loader. This results in a different hash. The canonical hash avoids these issues by transforming these instructions temporarily into a canonical form. A canonical hash is obtained by converting all instructions which might be changed during a relocation into a canonical form (e.g. based on load address 0) before calculating the hash. This feature suitable for use by a preferred embodiment of the present invention is described in patent application entitled "Integrity Checking An Executable Module And Associated Protected Service Provider Module", Application number 09/352,285, filed on 7/13/99, by International Business Machines, Inc., and is hereby incorporated by reference herein.

3. Another aspect of the invention is that the authentication token can be embedded in the external module as long as the external module itself remains functional. This has the advantage that the authentication token is always readily available when the external module is accessed. On the Windows family of operating systems, the embedding can be realized by adding an additional data section to a DLL in Portable Executable (PE) format. This does not affect the functionality of the DLL but allows for the data to be retrieved if necessary. The disclosure of the invention covers both external and embedded authentication tokens.

4. In addition to an initial authentication, the scheme presented in the disclosure can be further enhanced by performing run time checks to make sure that function calls to the external module are not intercepted by an attacker. For example, a breakpoint instruction (Trap x03 on the Intel Pentium) might be used to intercept the function calls before control is actually transferred to code in the external DLL.

5. In addition to an initial authentication, the external module can be re-authenticated periodically by performing the verification procedure described in the disclosure multiple times while interacting with an external module.

6. The process of creating the authentication token can be automated, for example by providing the developers of the external module access to a Web site which allows them to submit the hash of their module and to retrieve the authentication token. Of course the Web site would have to implement sufficient access control mechanisms in order to be accessible only by authorized third parties.

7. The transforming function is a stream encryption cipher in the UNSTOMP function, and a XOR with pseudo-random data in the STOMP. These two functions are logically equivalent, but generate quite different code. This complicates the hackers job of analyzing the STOMP/UNSTOMP behavior.

8. In a preferred embodiment, the key value K is expanded by using it as a seed to produce pseudo-random bytes. This is because software modules are generally much larger than the length of the value K that can be efficiently calculated with

RSA or signet technology, and the number of bytes that are needed for the STOMP/UNSTOMP behavior is equal to the size of the module. Of course, if the software module is small, or if the K calculation is more efficient, it is within the scope of this invention to forgo the pseudo-random bytes and use K directly.

Alternative Embodiments

An alternate embodiment envisions the use of signet data in both the STOMP and the UNSTOMP processes. Further, as a third option, the STOMP uses signets and the UNSTOMP uses another cryptographic technique like the RSA protocol to accomplish the authentication of the external module. All of these are within the scope of the invention.

Conclusion

Therefore, a more efficient method, apparatus and product for securing and verifying the authenticity of external modules has been described. Thus, the limitations defined in the prior art have been overcome. In addition, the description has included an improved system and method for controlling the enablement of an external module so that an application loading the external module fails in the event that the module is illicitly patched by a hacker. Otherwise, the actuation of the application is permitted because the STOMP-UNSTOMP procedure has validated the authenticity of the external module. Finally, the invention is implemented as part of the EMMS, or Electronic Media Management System End User Toolkit as of release 1.33. The invention is used to authenticate external decoder modules which are provided by third party companies.

Discussion of Hardware and Software Implementation Options

The present invention, as would be known to one of ordinary skill in the art could be produced in hardware or software, or in a combination of hardware and software. The system, or method, according to the inventive principles as disclosed in connection

with the preferred embodiment, may be produced in a single computer system having separate elements or means for performing the individual functions or steps described or claimed or one or more elements or means combining the performance of any of the functions or steps disclosed or claimed, or may be arranged in a distributed computer system, interconnected by any suitable means as would be known by one of ordinary skill in art.

According to the inventive principles as disclosed in connection with the preferred embodiment, the invention and the inventive principles are not limited to any particular kind of computer system but may be used with any general purpose computer, as would be known to one of ordinary skill in the art, arranged to perform the functions described and the method steps described. The operations of such a computer, as described above, may be according to a computer program contained on a medium for use in the operation or control of the computer, as would be known to one of ordinary skill in the art. The computer readable medium which may be used to hold, contain or deliver the computer program product, may be a fixture of the computer such as an embedded memory or may be on a transportable medium such as a disk, as would be known to one of ordinary skill in the art.

The invention is not limited to any particular computer program or logic or language, or instruction but may be practiced with any such suitable program, logic or language, or instructions as would be known to one of ordinary skill in the art. Without limiting the principles of the disclosed invention any such computing system can include, inter alia, at least a computer readable medium allowing a computer to read data, instructions, messages or message packets, and other computer readable information from the computer readable medium. The computer readable medium may include non-volatile memory, such as ROM, Flash memory, floppy disk, Disk drive memory, CD-ROM, and other permanent storage. Additionally, a computer readable medium may include, for example, volatile storage such as RAM, buffers, cache memory, and network circuits.

Furthermore, the computer readable medium may include computer readable information in a transitory state medium such as a network link and/or a network interface, including a wired network or a wireless network, that allow a computer to read such computer readable information.

Although specific embodiments of the invention have been disclosed, those having ordinary skill in the art will understand that changes can be made to the specific embodiments without departing from the spirit and scope of the invention. The scope of the invention is not to be restricted, therefore, to the specific embodiments, and it is intended that the appended claims cover any and all such applications, modifications, and embodiments within the scope of the present invention.

What is claimed is: